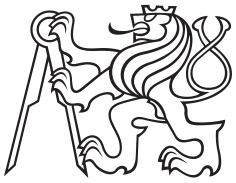
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Konvoluční neuronové sítě pro klasifikaci objektů z LiDARových dat

Jiří Zacha

Vedoucí: Ing. Patrik Vacek
Studijní program: Kybernetika a robotika
Květen 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Zacha** Jméno: **Jiří** Osobní číslo: **466276**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Konvoluční neuronové sítě pro klasifikaci objektů z LiDARových dat

Název bakalářské práce anglicky:

Convolutional Neural Networks for Object Classification from LiDAR Data

Pokyny pro vypracování:

Cílem této bakalářské práce je návrh a ověření Konvoluční neuronové sítě (CNN) pro klasifikaci LiDARových (Light Detection and Ranging) dat, které představují jedoucí automobily a dopravní situace. Data pochází z LIDARu typu Scala, který v porovnání s ostatními zařízeními snímá méně bodů v jednom měření. Vybraný dataset obsahující body měření bude použit k trénování a testování CNN modelu pro klasifikaci automobilů, chodců či jiných překážek, neboť tyto dedukované informace se dají dále využít v oblasti autonomní jízdy.

1. Navrhněte vhodnou reprezentaci LIDARových dat.
2. Sestavte architekturu CNN a natrénujte síť pro klasifikaci statických objektů ve scéně.
3. Otestujte model na testovacích datech a porovnejte různé navržené architektury.
4. Diskutujte možné přístupy klasifikování pohybujících se objektů.

Seznam doporučené literatury:

[1] Yang, Bin & Luo, Wenjie & Urtasun, Raquel. (2018). PIXOR: Real-time 3D Object Detection from Point Clouds. 7652-7660. CVPR 2018

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, available on http://www.deeplearningbook.org, 2016

[3] Fei-Fei Li, Justin Johnson, Serena Yeung, 2017, CS231n: Convolutional Neural Networks for Visual Recognition, video lectures, University Stanford, 2017, available through http://cs231n.stanford.edu/

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Patrik Vacek, vidění pro roboty a autonomní systémy FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **11.02.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Patrik Vacek	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Poděkování

Děkuji vedoucímu práce Ing. Patriku Vackovi za jeho rady a trpělivost při tvorbě této bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 24. května 2019

Abstrakt

Tato bakalářská práce se zabývá návrhem konvolučních neuronových sítí pro segmentaci dat z LiDARu Scala. Byla použita simulovaná data ze hry GTA V a reálná data od firmy Valeo v podobě point cloudů, které byly převedeny na hloubkové mapy a snímky z ptačí perpektivy. Predikce sítě byly použity k sémantické segmentaci bodů a následně k odhadu pohybu vozidel.

Klíčová slova: konvoluční neuronové sítě, LiDAR, segmentace

Vedoucí: Ing. Patrik Vacek
U12110.3

Abstract

This bachelor thesis focuses on designing a convolutional neural network for LiDAR data segmentation. The networks were trained using simulated data from GTA V and real data provided by Valeo represented by depth maps and BEV images. The predictions are then used to estimate motion of detected vehicles.

Keywords: convolutional neural networks, LiDAR, segmentation

Title translation: Convolutional Neural Networks for Object Classification from LiDAR Data

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Pro potřeby řízení autonomních aut potřebujeme vynikající porozumění okolní scény. Nemůžeme si dovolit chybovat, protože chyby můžou vést ke ztrátě lidských životů. Okolí auta je třeba snímat senzory a následně správně vyhodnotit. Používá se fúze více senzorů, nejčastěji kamer a radarů. V poslední době se začínají používat LiDARy, pro jejich schopnost vytvořit trojrozměrný obraz prostředí nezávisle na světelných podmínkách.
K zpracování dat se čím dál častěji používají konvoluční neuronové sítě, které mají v oblasti rozpoznávání aut, chodců a překážek nejlepší výsledky. Pro úspěšné trénování sítí však potřebujeme velké množství vstupních dat s anotovanými objekty. Tyto anotace v současnosti vytváří lidé, což je drahé a nepříliš efektivní.
V této práci se zaměříme na rozpoznání osobních a nákladních automobilů pomocí dat z LiDARu Scala vyvíjené firmou Valeo. Tento senzor vyniká nízkou cenou, která je ale vykoupena menším množstvím bodů, se kterými můžeme pracovat. Výstup tohoto senzoru jsme nejprve simulovali v počítačové hře *Grand Theft Auto V*, v konečné fázi práce jsme obrželi reálná data od firmy Valeo ve formě point cloudů. Vyzkoušíme více způsobů reprezentace těchto dat, jmenovitě hloubkové mapy a obrázky z ptačí perspektivy. Pro každou z těchto reprezentací navrhneme konvoluční síť, a jejich výsledky vyhodnotíme. Posledním bodem mé práce je odhad pohybu detekovaných vozidel napříč snímky. Námi navržené konvoluční neuronové sítě jsme implemetovali v jazyce Python za pomoci frameworku Pytorch.

Kapitola 2

Teorie

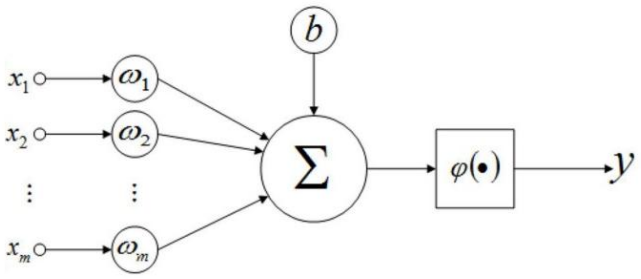
2.1 Základy neuronových sítí[1]

2.1.1 Model neuronu

Model neuronu svou strukturou a funkcí napodobuje biologický neuron. Nejčastěji používaný je model vycházející z práce McCullocha a Pittse *A logical calculus of the ideas immanent in nervous activity*[10]. Neuron je popsán vztahem:

$$y = \varphi(\boldsymbol{\omega}^T \mathbf{x} + b), \tag{2.1}$$

kde y je výstup neuronu, $\varphi(x)$ je aktivační funkce, $\mathbf{x}$ je vstupní vektor, $\boldsymbol{\omega}$ je vektor vah a b je bias, neboli práh.

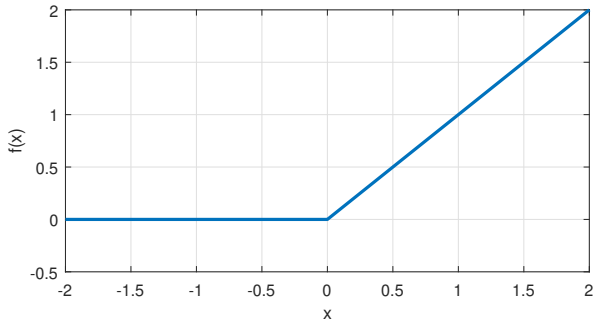


Obrázek 2.1: Model neuronu[2]

Aktivační funkce se používají k modelování nelinearity, umožňují tak neuronu vyhodnotit i komplexnější problémy. Aktivačních funkcí existuje mnoho druhů, nejčastěji se používají následující:

- ReLU (Rectified Linear Unit)

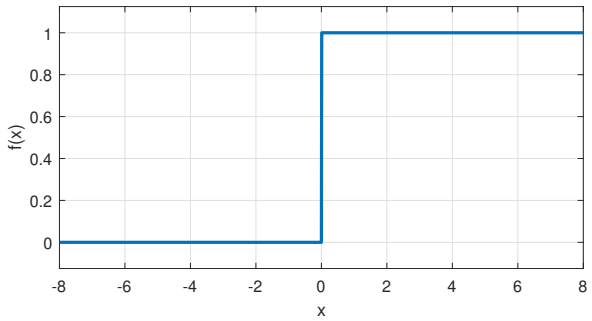
$$\varphi(x) = \max(0, x) \tag{2.2}$$



Obrázek 2.2: Průběh aktivační funkce ReLU

- skoková funkce

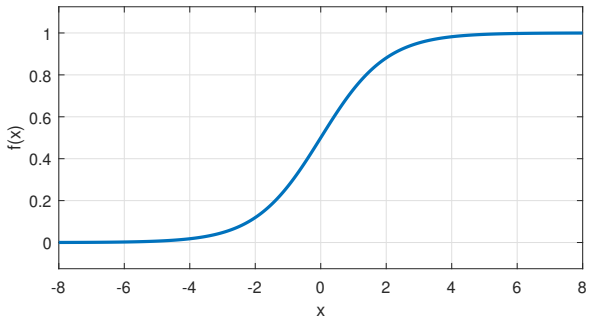
$$\varphi(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{2.3}$$



Obrázek 2.3: Průběh skokové aktivační funkce

- sigmoida

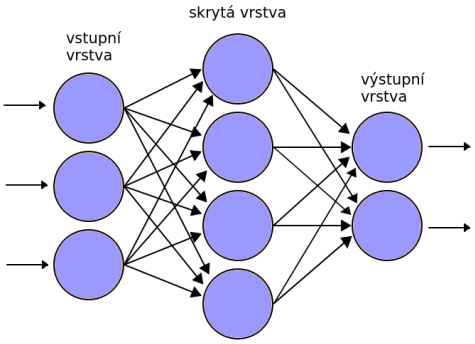
$$\varphi(x) = \frac{e^x}{1 + e^x} \tag{2.4}$$



Obrázek 2.4: Průběh aktivační funkce sigmoida

2.1.2 Neuronová síť

Propojením výstupů neuronů se vstupy dalších vzniká neuronová síť. Neurony jsou nejčastěji uspořádány v síti do několika vrstev, viz obrázek 2.5. Neurony náležící sousedícím vrstvám jsou propojeny. První vrstva, zvaná vstupní, obsahuje neurony s jediným vstupem. Neurony výstupní vrstvy naopak mají pouze jeden výstup. Zbylým vrstvám se říká skryté, jejich neurony jsou navzájem propojeny. Neuronová síť má schopnost učit se a přizpůsobovat vstupním datům. Činí tak změnou vnitřních parametrů $\boldsymbol{\omega}$ na základě úspěšnosti predikce. Obecné neuronové sítě nejsou příliš vhodné pro zpracování vysokodimenzionálních vstupů, které vyžadují velké množství vnitřních parametrů.



Obrázek 2.5: Schéma základní neuronové sítě[3]

2.2 Konvoluční neuronové sítě a jejich vrstvy[1]

Konvoluční neuronové sítě jsou podmnožinou neuronových sítí. Vyznačují se použitím konvolučních a pooling vrstev. Díky nim jsou schopny efektivně zpracovávat vstupy velkých rozměrů, jako jsou například obrázky, za použití mnohem menšího množství parametrů než obecná konvoluční síť. Díky tomu je snazší takovou síť naučit.
Konvoluční sítě se zpravidla skládají z několika různých typů vrstev. Základní typy, použité v praktické části, tu budou popsány.

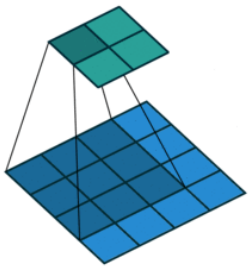
2.2.1 Konvoluční vrstva

Tato vrstva vykonává operaci konvoluce. Každá taková operace má definované konvoluční jádro, tedy matici nejčastěji o rozměrech 3×3 buněk. Dále je třeba znát velikost kroku a padding, neboli šířku rámu vstupní matice vyplněného nulami. Pro vstup o velikosti $m \times n$, jádro o rozměrech 3×3 jednotkový krok a nulový padding můžeme vyjádřit konvoluci následujícími vztahy:

$$\mathbf{Y} = \mathbf{X} * \mathbf{F} \tag{2.5}$$

$$y_{i,j} = \sum_{k=1}^{3} \sum_{l=1}^{3} f_{k,l}\, x_{i+k,j+l}, \tag{2.6}$$

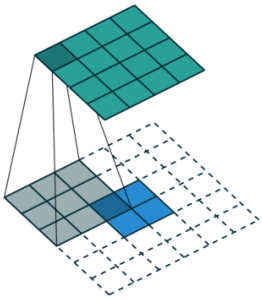
kde $\mathbf{Y}$ je výstupní matice, $\mathbf{X}$ vstupní matice, $\mathbf{F}$ jádro konvoluce, a $y_{i,j}$, $x_{i,j}$, $f_{i,j}$ jsou prvky matic $\mathbf{Y}$, $\mathbf{X}$ a $\mathbf{F}$.
Obecně operaci provádíme tak, že jádro postupně přikládáme na vstupní matici, poté pronásobíme na sobě ležící prvky a jejich součet zapíšeme do výstupní matice. Poté posuneme jádro o daný krok a pokračujeme. Výhodou konvolučních vrstev je schopnost rozeznat specifické příznaky při zachované informaci o jejich poloze. Často se také konvoluční vrstvy využívají k zvýšení počtu kanálů výstupu (3. rozměru výstupní matice) pro lepší přenos informace. Na obrázku 2.6 jsou vstupy konvoluce vyvedeny modrou barvou, výstupy zelenou.



Obrázek 2.6: Znázornění operace konvoluce[4]

2.2.2 Dekonvoluční vrstva

Dekonvoluční vrstva, někdy také transponovaná konvoluční vrstva, provádí inverzní operaci ke konvoluci, tedy se snaží odhadnout vstupní matici konvoluce ze znalosti matice výstupní. Na obrázku 2.7 je tato operace znázorněna. Vstupní matice je označena modrou barvou, výstupní zelenou.

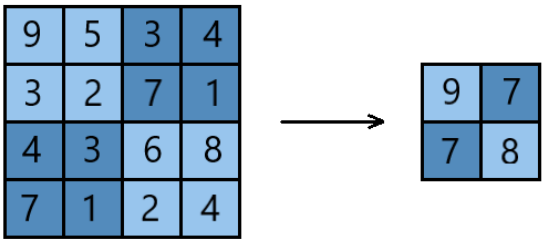


Obrázek 2.7: Znázornění operace dekonvoluce[4]

2.2.3 Maxpool vrstva

Maxpool vrstva slouží k snížení velikosti vstupů a tedy nepřímo i k zlepšení distribuce informace. Dále snižuje výpočetní náročnost učení. Každá vrstva má definovanou velikost jádra, tedy velikost podoblasti vstupní matice. Dále definujeme horizontální krok h a vertikální krok v, o které se podoblast posouvá vůči vstupní matici. Nejčastěji se setkáme s velikostí jádra 2×2 a kroky o stejné velikosti jako je šířka jádra, tedy 2. Vrsta vybere maximální hodnotu dané podoblasti a zapíše ji na výstup, poté se posune dále. Pro vstup o velikosti $m \times n$ dostaneme výstup o velikosti $\frac{m}{v} \times \frac{n}{h}$.

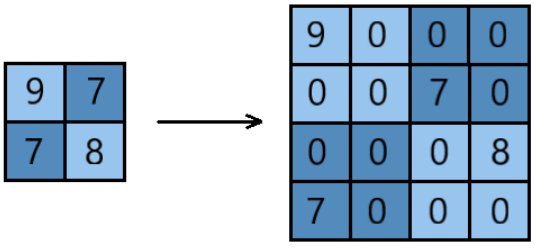
V případě že budeme chtít provádět opačnou operaci unpooling, je třeba si pro každou výstupní hodnotu zapamatovat i její polohu v rámci její podoblasti.



Obrázek 2.8: Znázornění operace maxpooling

2.2.4 Unpool vrstva

Unpool vrstva provádí operaci inverzní k operaci maxpooling, tedy slouží k opětovnému zvýšení velikosti vstupů. Stejně jako operace maxpooling má definovanou velikost jádra a kroku v, h, které by se měly shodovat s hodnotami použitými v maxpool vrstvě. Kromě vstupní matice potřebuje vrstva navíc ještě matici poloh maxim v rámci jádra. Pro vstupní matici o velikosti $m \times n$ dostaneme výstup o velikosti $v\,m \times h\,n$.
Na začátku si vytvoříme matici nul o velikosti výstupní matice. Poté každý prvek vstupní matice zapíšeme do odpovídající podoblasti výstupní matice na relativní pozici danou maticí poloh maxim. V případě překryvu dvou hodnot můžeme vybrat maximální z hodnot, ale protože nejčastěji volíme velikost kroku shodnou s velikostí jádra, k žádným překryvům nedochází.



Obrázek 2.9: Znázornění operace unpooling

2.2.5 Fully connected vrstva

Fully connected (FC) vrstva, někdy také nazývána lineární nebo dense vrstva, vytváří spojení mezi každým vstupním prvkem a každým výstupním prvkem. Toto spojení napomáhá distribuci informace z celého vstupu. Pro každou vrstvu je třeba definovat rozměr vstupu a výstupu. Vrstva očekává na vstupu vektor, takže musíme případnou vstupní matici vektorizovat. Samotná operace provádí afinní zobrazení vstupu na výstup, které můžeme vyjádřit vztahem

$$y = \mathbf{A}^T\mathbf{x} + \mathbf{b}\,, \tag{2.7}$$

kde $\mathbf{x}$ je vstupní vektor, $\mathbf{y}$ je výstupní vektor, $\mathbf{A}$ je transformační matice a $\mathbf{b}$ je vektor posunu. Nevýhodou této operace je velký počet parametrů, který je pro vstupní vektor o délce i a výstupní vektor o délce o roven $o(i + 1)$.

2.3 Učení konvolučních sítí

Učení neuronové sítě je stejně jako její koncepce inspirováno člověkem. Stejně jako se dítě učí rozpoznávat a pojmenovávat předměty kolem sebe za pomoci rodičů, kteří mu poskytují zpětnou vazbu, tak i síť dostává informace o úspěšnosti své predikce. Cílem učení neuronové sítě je tedy optimalizace jejích vnitřních parametrů s cílem minimalizovat chybu predikce. Fáze učení se skládá z mnoha cyklů, neboli iterací, obsahujících dopřednou propagaci, vyhodnocení chyby predikce, zpětnou propagaci a optimalizace parametrů.

2.3.1 Dopředná propagace

Při dopředné propagaci (foward pass) projde vstupní matice všemi vrstvami na výstup. Jeho rozměr je závislý na funkci dané sítě. V našem případě se jedná o segmentaci, tedy zařazení každého bodu vstupní matice do jedné z k kategorií. Rozměr výstupu tedy musí být shodný s rozměrem vstupu a mít k kanálů. Hodnoty v jednotlivých buňkách jsou v rozmezí 0 až 1 a určují míru důvěry sítě ve správnost klasifikace dané kategorie pro vybraný prvek vstupní matice. Dále musí platit, že součty hodnot v buňkách jsou rovny 1, čehož dosáhneme aplikací funkce softmax na výstup poslední vrstvy sítě.

2.3.2 Ztrátová funkce

Ztrátová funkce určuje míru rozdílu predikce a referenčních hodnot, neboli ground truth (GT). Mezi nejčastější z nich patří:

- L1 loss

$$L = l(\mathbf{P}, \mathbf{GT}) = \sum_{i=1}^{m}\sum_{j=1}^{n} |\mathbf{p_{i,j}} - \mathbf{gt_{i,j}}| \tag{2.8}$$

- L2 loss

$$L = l(\mathbf{P}, \mathbf{GT}) = \sum_{i=1}^{m}\sum_{j=1}^{n} (\mathbf{p_{i,j}} - \mathbf{gt_{i,j}})^2 \tag{2.9}$$

- Cross entropy loss

$$L = l(\mathbf{P}, \mathbf{GT}) = \sum_{i=1}^{m}\sum_{j=1}^{n} (-p_{i,j,gt_{i,j}} + \log(\sum_{class} \exp(p_{i,j,class}))) \tag{2.10}$$

Ztrátové funkce navíc umožňují přidat váhy jednotlivých kategorií pro kompenzaci nevyvážených trénovacích množin. Dále je možné označit prvky, které se nepodílejí na výsledné chybě.

2.3.3 Zpětná propagace

Po určení chyby provedeme její zpětnou propagaci (backpropagation), tedy určíme podíl jednotlivých vah sítě na výsledné chybě. Mezi váhy sítě počítáme jádra konvolucí a dekonvolucí, matice a vektory afinního zobrazení lineárních vrstev, případně všechny další parametry, jež má smysl učit. Pro každý parametr určíme parciální derivaci chyby podle řečeného parametru, kterou následně vyčíslíme.

2.3.4 Optimalizace parametrů

Následně je třeba řešit problém optimalizace parametrů, jehož cílem je snížit hodnotu chyby. K tomu se nejčastěji využívá některé z gradientních metod. V prostoru všech parametrů sítě najdeme gradient ztrátové funkce, jehož složky

jsme vypočítali zpětnou propagací. Poté vykonáme krok ze stávajícího vektoru parametrů ve směru opačném ke směru největšího růstu, tedy gradientu. Velikost kroku ϵ je předem definovaná, někdy se také nazývá rychlost učení. Nový vektor bude mít podobu:

$$\mathbf{p}_{i+1} = \mathbf{p}_i - \epsilon \nabla L \,, \tag{2.11}$$

kde $\mathbf{p}_i$ je stávající vektor parametrů, $\mathbf{p}_{i+1}$ je nový vektor parametrů, ϵ velikost kroku a L ztrátová funkce. Současný algoritmus můžeme vylepšit přidáním setrvačnosti α, která napomáhá algoritmu vyvarovat se lokálních minim ztrátové funkce. Nový vztah pro aktualizaci parametru má tvar:

$$\mathbf{p}_{i+1} = \mathbf{p}_i - \epsilon \nabla L + \alpha(\mathbf{p}_i - \mathbf{p}_{i-1}) \tag{2.12}$$

Další vylepšení nabízí například algoritmus Adam (ADAptive Moment estimation)[11], který průběžně upravuje velikost kroku ϵ a setrvačnost α, které jsou navíc unikátní pro každý parametr.

2.3.5 Rozdělení vstupních dat

Tradičně máme vstupní data rozdělená do dvou množin, trénovací a testovací, neboli validační. Trénovací množina slouží k samotnému učení sítě, testovací množina by měla reprezentovat reálná data, na která se síť nenaučila. V jednotlivých iteracích tak běžně počítáme chybu na trénovací i na testovací množině, při počítání chyby na testovací množině ale vynecháme fázi zpětné propagace a optimalizace parametrů. Chyba testovací množiny by měla být ukazatelem schopnosti sítě generalizovat.

2.4 Nástrahy učení

Protože má běžně prostor parametrů sítě vysokou dimenzi, úloha hledání globálního minima nelineární funkce na tomto prostoru je velice obtížná. Optimalizační algoritmy často naleznou lokální minimum, ze kterého se už nedostanou. Algoritmy používají různé metody jak se těmto lokálním minimům vyhnout, nebo se z nich dostat, například již dříve zmíněná setrvačnost.

2.4.1 Přeučení

Další komplikaci představuje neúplná informace o průběhu učení. Jediné veličiny, které nám tuto informaci poskytují, jsou trénovací a testovací chyba. V ideálním případě by obě tyto veličiny současně konvergovaly k nule, ve skutečnosti ale ve fázi učení pravděpodobně nastane chvíle, kdy se klesající trend testovací chyby zastaví a chyba začne opět stoupat, zatímco trénovací chyba dále klesá. Tento jev se nazývá přeučení, neboli overfitting, kde se síť naučí rozeznávat příznaky, které jsou obsaženy pouze v prvcích trénovací množiny.

2.4.2 Předčasné ukončení

Existuje několik způsobů jak s tímto jevem bojovat. Nejjednodušším z nich je předčasné zastavení učení. Ve chvíli, kdy začíná testovací chyba růst, zastavíme učení a uložíme parametry sítě. Protože není vyloučené, že testovací chyba začne opět klesat, můžeme určit záchytné body po určitém počtu iterací, kde uložíme parametry sítě a následně vybrat ty s nejlepšími výsledky.

2.4.3 Normalizace dat

Dalším způsobem jak omezit přeučení sítě je normalizace dat. Normalizace dat pomáhá kompenzovat rozdíly v rozsazích hodnot jednotlivých prvků vstupních dat. Pokud totiž mají přispívat všechny prvky stejnou vahou na výsledek predikce, musí být tyto rozdíly kompenzovány velikostí odpovídajících parametrů sítě, které budou mít větší vliv na vývoj gradientu a tedy i na optimalizaci parametrů sítě. Tento princip ale můžeme uplatnit na vstupy každé vrstvy a čímž dosáhneme efektivnějšího učení.

2.4.4 Dropout

V neposlední řadě s přeučením bojuje Dropout. V průběhu učení přerušuje spojení mezi některými neurony dle náhodného výběru s předem určenou pravděpodobností. Díky tomu se průběžně mění váhy, přispívající do ztrátové funkce, což by mělo rovnoměrněji rozložit vliv vah, ovlivňujících výsledek predikce.

2.5 Metoda vyhodnocení

Abychom mohli vyhodnotit úspěšnost klasifikace, potřebujeme stanovit objektivní metriku. Nejčastěji se k tomuto účelu používá matice záměn (confusion matrix). Jedná se o čtvercovou matici, jejíž rozměr odpovídá počtu klasifikovaných kategorií, a její prvky ukazují vztah predikcí a správných hodnot. V jejích řádcích najdeme výsledky predikce, v sloupcích referenční hodnoty. Na diagonále se tedy nachází počty správných predikcí pro každou kategorii.

		Reference		
		a	**b**	**c**
Predikce	**a**	5	2	0
	b	3	3	2
	c	0	1	11

Tabulka 2.1: Ukázka matice záměn pro 3 kategorie, **a, b, c**

Dále pro každou kategorii definujeme veličiny precision a recall. Precision udává poměr správně klasifikovaných ku všem klasifikacím, recall popisuje, kolik z referenčních hodnot klasifikátor určil správně. Tyto hodnoty jsou obecně definovány pro N kategorií a matici záměn $\mathbf{M}$ následujícími vztahy:

$$precision_i = \frac{\boldsymbol{M}_{i,i}}{\sum_{j=1}^{N} \boldsymbol{M}_{i,j}} \tag{2.13}$$

$$recall_i = \frac{\boldsymbol{M}_{i,i}}{\sum_{j=1}^{N} \boldsymbol{M}_{j,i}} \tag{2.14}$$

2.6 Nástroje pro učení konvolučních sítí

Existuje mnoho nástrojů určených k práci s konvolučními sítěmi. Nejčastěji mají podobu knihoven programovacích jazyků, například pro Python existují Pytorch, TensorFlow a Keras, pro C++ se používá Caffe. Tyto nástroje jsou připraveny na zpracování matic mnoha rozměrů a obsahují implementace jednotlivých vrstev a prostředků potřebných k učení. Navíc často nabízí

možnost spuštění na grafických kartách, což urychlí běh programu. V této práci byl použit framework Pytorch pro svou jednoduchost a přehlednost.

Kapitola 3

Segmentace LiDARových dat

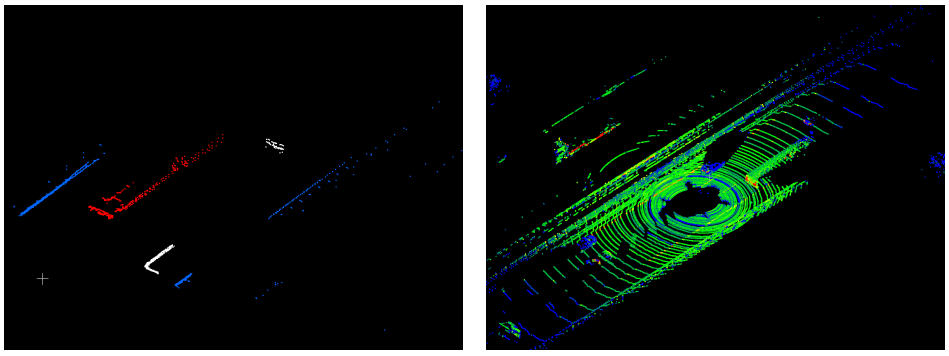
3.1 Senzor Scala

Scala je senzor vzdálenosti typu LiDAR (Light Detection And Ranging) vyvíjený firmou Valeo. Senzory tohoto typu pracují na podobném principu a skládají se nejčastěji z vysílače, přijímače a otočné základny. Vysílač vyšle laserový puls, přijímač čeká na jeho odraz a měří dobu letu. Z té je pak schopen určit vzdálenost objektu. Poté se otočná základna otočí kolem svislé osy o předem definovaný úhel, někdy nazývaný úhlové rozlišení, a vyšle další puls. Tento základní model senzoru ale skenuje okolní prostor pouze v jedné rovině. Protože ale ve většině aplikací potřebujeme více informací o okolní scéně, mnoho komerčních LiDARů používá těchto vrstev několik, aby byla pokryta větší část prostoru. Scala má vrstvy 4 s úhlovým rozestupem o velikosti 0.8°, konkurenční senzory značky Velodyne mají vrstev 16, 32, nebo až 64. Další důležitý údaj je zorný úhel senzoru, který je v případě Scaly 145° s úhlovým rozlišením 0.25°, a maximální dosah 80 m. Celkem tedy můžeme v jednom měření naměřit až 2320 bodů, většinou ale přijdeme o třetinu až polovinu tohoto počtu, protože se paprsek vůbec neodrazí, nebo se odrazí od lesklého povrchu a nevrátí se zpět k senzoru.

	Valeo Scala	Velodyne HDL-64E
Horizontální zorný úhel	145°	360°
Horizontální úhlové rozlišení	0.25°	0.09°
Vertikální zorný úhel	3.2°	26.8°
Vertikální úhlové rozlišení	0.8°	≈ 0.4°
Počet vrstev	4	64
Počet měření za sekundu	10-25	5-15
Dosah	80 m	120 m
Přesnost měření	10 cm	<2 cm
Cena	4000 $	100000 $

Tabulka 3.1: Porovnání LiDARů Scala[8] a HDL-64E[9]

Obrázek 3.1: LiDAR Scala vyvíjený firmou Valeo[5]



(a) : Vizualizace dat z LiDARu Valeo Scala

(b) : Vizualizace dat z LiDARu Velodyne HDL-64E

Obrázek 3.2: Porovnání počtu bodů výstupu LiDARů Scala a HDL-64E v programu CloudCompare[6]

Na obrázku 3.2a vidíme červenou barvou označené body náležící projíždějícímu kamionu, zeleně označená auta a modře ostatní objekty, což jsou v tomto případě svodidla lemující silnici. Na obrázku 3.2b znázorňují barvy intenzitu odražených paprsků. Oba senzory snímají stejnou scénu.

3.2 Vstupní data

Protože je senzor Scala stále ve fázi vývoje, nelze jeho data nalézt v žádném z veřejných datasetů. Jednou z motivací této práce byl příslib reálných dat od firmy Valeo, které jsme ale obdrželi až v konečné fázi práce.

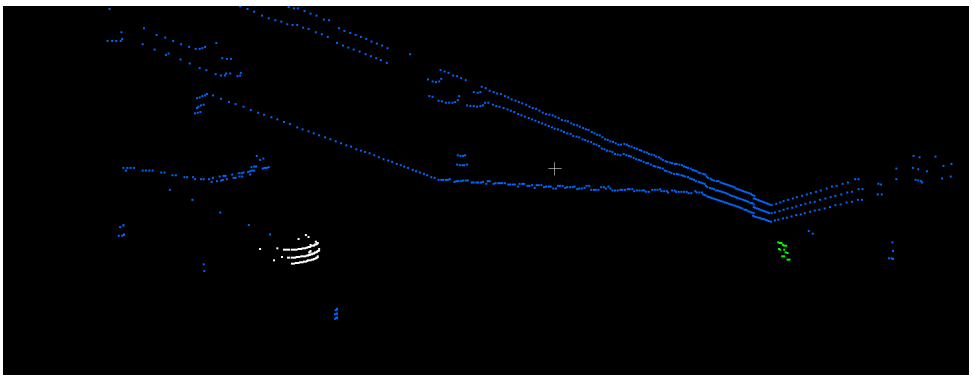
3.2.1 Simulovaná data

Pro návrh sítě jsme proto použili simulovaná data získaná ve formě anotovaných RGBD obrázků. Ty byly nasnímány ve hře Grand Theft Auto V od firmy Rockstar Games za pomoci rozšíření umožňujícího ovlivňovat okolní scénu. Obrázky byly nasnímány kamerami umístěnými na střeše auta ve čtyřech směrech. Tato data obsahují anotace pro 4 kategorie: pozadí, auta, lidi a vegetaci, my jsme se však omezili jen na první 3 a body náležící vegetaci přiřadili k pozadí. Velká část těchto dat je nasbírána ve městě s nízkým počtem aut. Protože se jedná o uměle vytvořená data, můžou obsahovat artefakty, které se u reálných dat nevyskytují. Dále v nich není přítomen šum vlastní pro všechna reálně měřená data. Nejprve jsme měli k dispozici pouze hloubkové obrázky, z nichž byl vytvořen naivní model výstupu senzoru. Tato data sloužila převážně k zodpovězení otázky, zda-li se může síť úspěšně naučit segmentaci na tak malém množství vstupních bodů.

Obrázek 3.3: Příklad scény ze hry GTA V s označenými kategoriemi

Na obrázku 3.3 můžeme vidět příklad příklad scény ve hře Grand Theft Auto V. Ve střední části obrázku se nachází 4 linie značící roviny snímání senzoru s barevně vyznačenými anotacemi.
V další fázi práce byl vytvořen přesnější model vycházející z geometrie senzoru, snažící se z v hloubkových mapách najít prvky odpovídající paprskům senzoru a z nich následně vytvořit point cloud. Tato data navíc můžou obsahovat aproximační chyby. Z těchto dat jsme vytvořili obrázky z pohledu

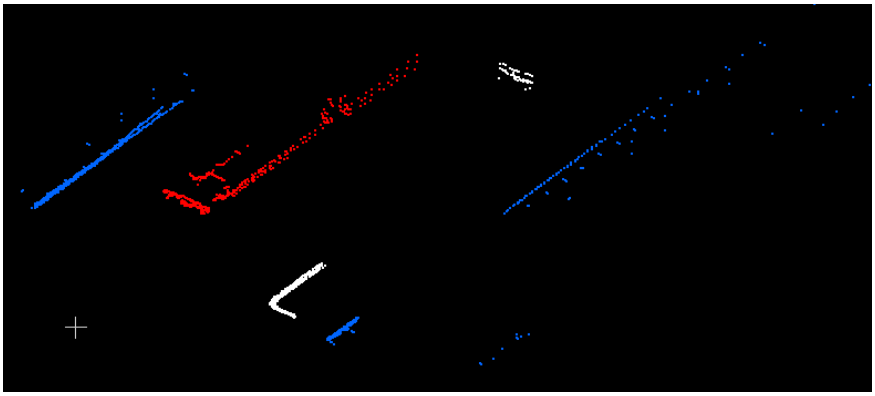
ptačí perspektivy.



Obrázek 3.4: Vizualizace stejné scény ze hry GTA V programem CloudCompare

3.2.2 Reálná data

V konečné fázi jsme obdrželi reálná data od firmy Valeo. V těchto datech je přítomen šum a obsahují anotace pro 4 kategorie: pozadí, osobní automobily, nákladní automobily a motocykly. Z důvodu velmi nízkého výskytu motocyklů jsme tuto kategorii spojili s kategorií osobních automobilů. Tato data obsahují převážně dálnice a rychlostní silnice.



Obrázek 3.5: Vizualizace reálné scény programem CloudCompare

Data jsme obdrželi v několika skupinách, z nichž každá obsahovala sekvenci měření. Každou skupinu jsme rozdělili v poměru 1:3 mezi testovací a trénovací množinu se zachovaným pořadím měření. Celkem jsme použili přibližně 6500 snímků, z toho asi 4800 v trénovací množině.

3.3 Hloubkové mapy

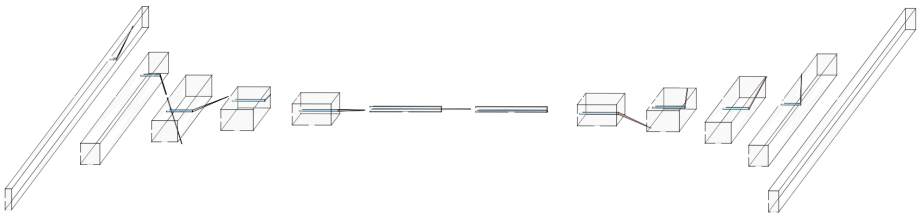
Nejpřímější způsob reprezentace dat z LiDARů jsou hloubkové mapy. V našem případě se jedná o matice o rozměru 4×580, jejíž prvky obsahují vzdálenosti objektů v odpovídajícím směru. K dispozici jsme měli data získaná ze hry GTA V, která jsme upravili tak, aby odpovídala výstupu LiDARu Scala. Způsob úpravy dat použitý v této části je velmi naivní, pouze jsme vybrali 4 řádky hloubkového obrázku a z každého řádku 580 bodů. Tak vznikly vstupní hloubkové mapy o rozměru 4×580.

3.4 Použitá architektura sítě

Protože řešíme problém segmentace, tedy klasifikace každého bodu vstupního obrázku, musí být rozměr vstupního a výstupního obrázku shodný. Při návrhu architektury sítě jsme se inspirovali prací *Learning Deconvolution Network for Semantic Segmentation*[12], ve které je použita síť ve tvaru přesýpacích hodin. Nejprve se maxpool vrstvami zmenšuje rozměr map a konvolučními vrstvami zvyšuje se počet kanálů. Množství přenášených informací se tak zmenšuje, což nutí síť přenášet pouze ty nejdůležitější informace. V polovině sítě se nachází 2 fully connected vrstvy, po nichž následuje zvyšování rozměrů unpool vrstvami a snižování počtu kanálů dekonvolučními vrstvami. Výstup má tedy rozměr shodný se vstupem s rozdílem počtu kanálů, který u výstupu odpovídá počtu rozpoznávaných kategorií. Celkem má síť 5 dvojic konvoluce-maxpool a dekonvoluce-unpool. Pooling vrsty mají jádra o netradičním rozměru 1×2 z důvodu malé výšky vstupní matice. Na výstupu každé konvoluční a dekonvoluční vrstvy se nachází ještě ReLU a dropout vrstvy, které jsou však v tabulce 3.2 vynechány pro větší přehlednost.

Název	Jádro	Krok	Padding	Rozměr výstupu
vstup				$4 \times 580 \times 1$
conv1	3×3	1	1	$4 \times 580 \times 4$
maxpool	1×2	2	0	$4 \times 290 \times 4$
conv2	3×3	1	1	$4 \times 290 \times 16$
maxpool	1×2	2	0	$4 \times 145 \times 16$
conv3	3×3	1	1	$4 \times 145 \times 32$
maxpool	1×2	2	0	$4 \times 72 \times 32$
conv4	3×3	1	1	$4 \times 72 \times 64$
maxpool	1×2	2	0	$4 \times 36 \times 64$
conv5	3×3	1	1	$4 \times 36 \times 128$
maxpool	1×2	2	0	$4 \times 18 \times 128$
přeskládání do vektoru				4*18*128
fc1				2048
fc2				4*18*128
přeskládání do matice				$4 \times 18 \times 128$
unpool	1×2	2	0	$4 \times 36 \times 128$
deconv5	3×3	1	1	$4 \times 36 \times 64$
unpool	1×2	2	0	$4 \times 72 \times 64$
deconv4	3×3	1	1	$4 \times 72 \times 32$
unpool	1×2	2	0	$4 \times 145 \times 32$
deconv3	3×3	1	1	$4 \times 145 \times 16$
unpool	1×2	2	0	$4 \times 290 \times 16$
deconv2	3×3	1	1	$4 \times 290 \times 8$
unpool	1×2	2	0	$4 \times 580 \times 8$
deconv1	3×3	1	1	$4 \times 580 \times$ počet kategorií

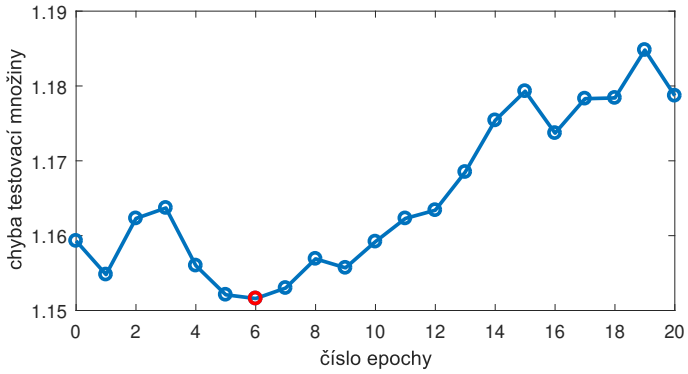
Tabulka 3.2: Popis výsledné architektury sítě



Obrázek 3.6: Architektura výsledné sítě[7]

3.4.1 Průběh učení

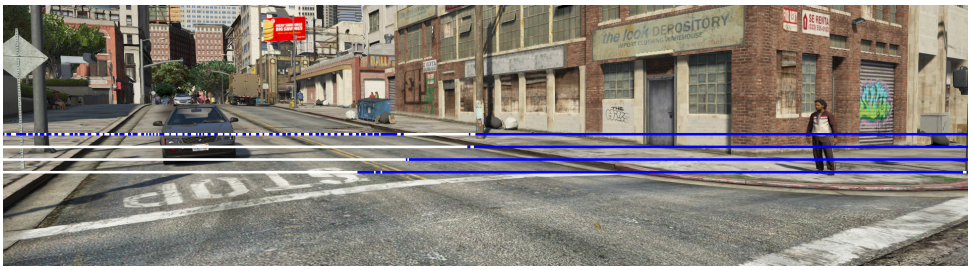
Síť jsme učili ve 20 epochách po 500 iteracích, v každé iteraci zpracovala síť 32 náhodně vybraných měření. Po každé epoše jsme určili chybu na celé testovací množině a uložili váhy sítě. Použili jsme ztrátovou funkci Cross entropy loss a optimalizační algoritmus Adam. Před začátkem učení jsme inicializovali váhy Xavier inicializací[13]. Na obrázku 3.7 vidíme průběh chyby testovací množiny. Ta se příliš nezmenšuje, naopak po 6. iteraci opět začíná růst. To svědčí o špatném učení sítě. Pro testování jsme vybrali váhy sítě po 6. iteraci s minimální chybou, v grafu vyznačené červeně.



Obrázek 3.7: Graf průběhu testovací chyby

3.4.2 Výsledky segmentace

Síť se nebyla schopna dostatečně naučit rozpoznávat vstupní data, o čemž svědčí vizulizace výstupu i matice záměn. Na obrázku 3.8 vidíme, že je síť schopna přibližně určit stojící auto, ale velmi nepřesně, označí při tom i velkou část jeho okolí náležící pozadí. Člověka na pravé straně obrázku síť ignoruje.



Obrázek 3.8: Příklad scény ze hry GTA V s označenými kategoriemi

Špatné výsledky predikce popisuje i matice záměn a níže vyčíslené veličiny precision a recall. Hodnoty pro auto navíc potvrzují naši domněnku, že je síť schopna přibližně auto rozeznat, ale s nízkou přesností.

		Reference		
		pozadí	**auto**	**nákl. auto**
Predikce	**pozadí**	448928	6515	23246
	člověk	156	1	7
	auto	224897	3924	11526

Tabulka 3.3: Matice záměn pro segmentaci 3 kategorií

	precision	**recall**
pozadí	93.78%	66.6%
člověk	0.61%	0.0%
auto	4.80%	33.1%

Tabulka 3.4: Tabulka hodnot precision a recall pro 3 kategorie

3.4.3 Zhodnocení

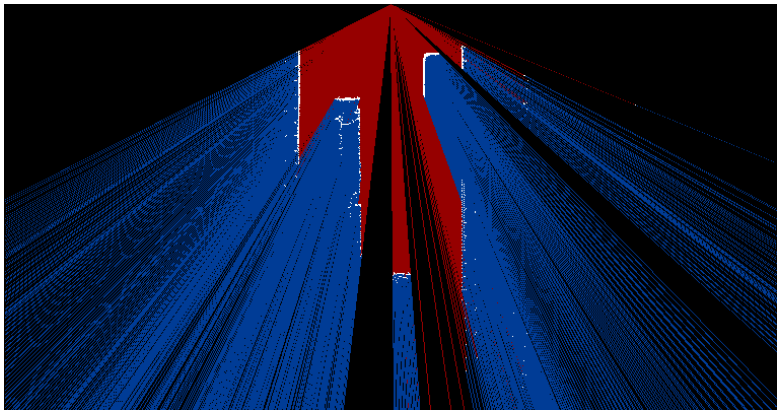
Tento přístup měl mnoho problémů. Prvním jsou vstupní data. Jedná se o složité scény obsahující málo aut a ještě méně lidí, na kterých by se mohla síť naučit. Další problém vychází z konstrukce senzoru. Paprsky LiDARu vycházejí ze stejného bodu, z toho důvodu jsou vzdálenější objekty určeny mnohem menším počtem bodů než ty blízké. Protože data nijak netransformujeme, velikost objektů je závislá na jejich vzdálenosti. To ale ztěžuje učení, protože síť musí tyto zdánlivě rozdílné objekty zařadit do stejné kategorie. Navíc v případě objektů v bezprostřední blízkosti senzoru, které obsadí značnou část vstupní hloubkové mapy, se nemusí informace o celém objektu zpropagovat na výstup sítě. Ukázali jsme ale, že i s malým množstvím má smysl provádět segmentaci.

3.5 Ptačí perspektiva

Problémy hloubkových dat odstraňuje projekce point cloudů do obrázků z ptačí perspektivy (bird's-eye view). Na nich se s rostoucí vzdáleností od senzoru nemění velikost objektu, ale pouze hustota bodů, které jej popisují. Dále mají vytvořené obrázky běžnější rozměr a poměr stran, což nám umožní se inspirovat existujícími architekturami.[14] Paralelně jsme vytvořili sítě a data pro segmentaci 2 a 3 kategorií.

3.5.1 Úprava dat

Vstupní data jsme dostali ve formě point cloudu. S těmi ale síť neumí pracovat, proto bylo třeba vygenerovat vstupní matice. Ty získáme promítnutím point cloudu na rovinu danou osami x, y a následným zarovnáním do buněk předem definované mřížky. Pro mřížku jsme určili následující parametry: požadované rozlišení je 10 cm, minimální vzdálenost kraje od senzoru 40 m, zorný úhel senzoru je 145°. Velikost mřížky jsme tedy určili jako minimální obdélník pokrývající kruhovou výseč o poloměru 40 m a úhlem 145°, výsledný rozměr je 762×400 prvků. Dále bylo třeba co nejlépe vyfiltrovat body silnice, čehož jsme dosáhli omezením rozsahu výšek bodů na interval $(0.1\,m, 3\,m)$.
V této chvíli by bylo možné použít jako vstupní data matice s binární informací o přítomnosti naměřeného bodu, ale tím bychom přišli o velké množství informací, které jsou obsaženy v point cloudu. Jednou z nich je informace o výšce naměřených bodů. V případě, že se více naměřených bodů promítne do jedné buňky mřížky, což může nastat, například když paprsky narazí na svislou stěnu budovy, musíme určit jedinou hodnotu, která danou buňku bude reprezentovat. Možností je více, například rozptyl těchto hodnot, nebo maximální hodnota. V našich vstupních datech jsme se rozhodli pro maximální hodnotu. Následně každé měření normujeme a abychom snížili paměťovou náročnost, zobrazíme je na interval 0-255 a hodnoty uložíme jako 8-bitové integery bez znaménka. Buňky do nichž se žádný bod nepromítl necháme nulové.
Další cenná informace plynoucí z principu funkce senzoru je informace o průchodnosti prostoru. Víme totiž, že pokud paprsek narazil na nějaký objekt a znovu se od něj odrazil, cesta k tomuto objektu je volná. Naopak můžeme předpokládat, že za nalezenou hranicí objektu je prostor neprůchozí. Můžeme tedy ve vstupní matici odlišit několik stavů: nalezený objekt označíme 1, průchozí prostor označíme 2, neprůchozí 3 a protože o zbylých buňkách nemáme žádné informace, zapíšeme do nich 0.



Obrázek 3.9: Vizualizace matice průchodnosti prostředí

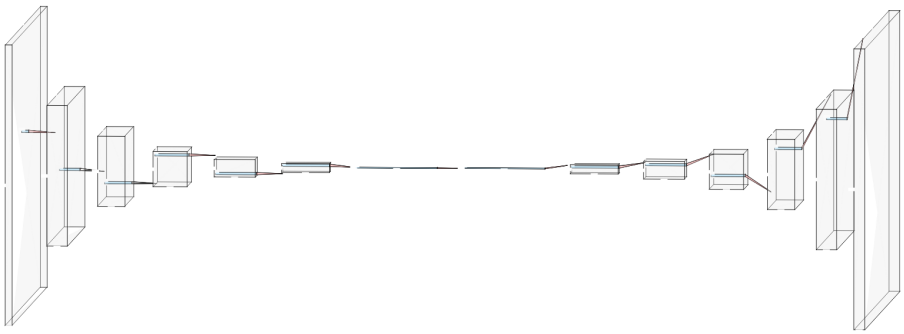
3.5.2 Výsledná architektura sítě

Architektura sítě má podobnou podobu jako síť v předchozí části. Liší se vyšším počtem kanálů a hloubkou, obsahuje totiž 6 dvojic konvoluce-maxpool a dekonvoluce-unpool. Pooling vrstvy mají v tomto případě jádro o rozměru 2×2.
Po několika obměnách jsme dospěli k architektuře, popsané v tabulce 3.5. Za každou konvoluční vrstvou následují ještě ReLU a dropout vrstvy, které jsou pro přehlednost v tabulce vynechány.

Název	Jádro	Krok	Padding	Rozměr výstupu
vstup				$400 \times 762 \times 2$
conv1	3×3	1	1	$400 \times 762 \times 8$
maxpool	2×2	2	0	$200 \times 381 \times 8$
conv2	3×3	1	1	$200 \times 381 \times 16$
maxpool	2×2	2	0	$100 \times 190 \times 16$
conv3	3×3	1	1	$100 \times 190 \times 32$
maxpool	2×2	2	0	$50 \times 95 \times 32$
conv4	3×3	1	1	$50 \times 95 \times 64$
maxpool	2×2	2	0	$25 \times 47 \times 64$
conv5	3×3	1	1	$25 \times 47 \times 128$
maxpool	2×2	2	0	$12 \times 23 \times 128$
conv6	3×3	1	1	$12 \times 23 \times 256$
maxpool	2×2	2	0	$6 \times 11 \times 256$
přeskládání do vektoru				6*11*256
fc1				4096
fc2				6*11*256
přeskládání do matice				$6 \times 11 \times 256$
unpool	2×2	2	0	$12 \times 23 \times 256$
deconv6	3×3	1	1	$12 \times 23 \times 128$
unpool	2×2	2	0	$25 \times 47 \times 128$
deconv5	3×3	1	1	$25 \times 47 \times 64$
unpool	2×2	2	0	$50 \times 95 \times 64$
deconv4	3×3	1	1	$50 \times 95 \times 32$
unpool	2×2	2	0	$100 \times 190 \times 32$
deconv3	3×3	1	1	$100 \times 190 \times 16$
unpool	2×2	2	0	$200 \times 381 \times 16$
deconv2	3×3	1	1	$200 \times 381 \times 8$
unpool	2×2	2	0	$400 \times 762 \times 8$
deconv1	3×3	1	1	$400 \times 762 \times$ počet kategorií

Tabulka 3.5: Popis výsledné architektury sítě

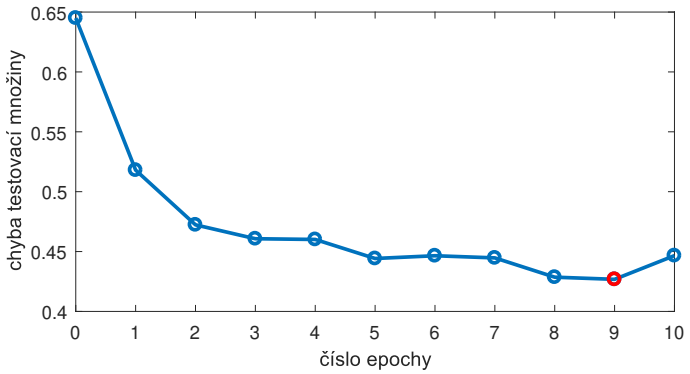
.



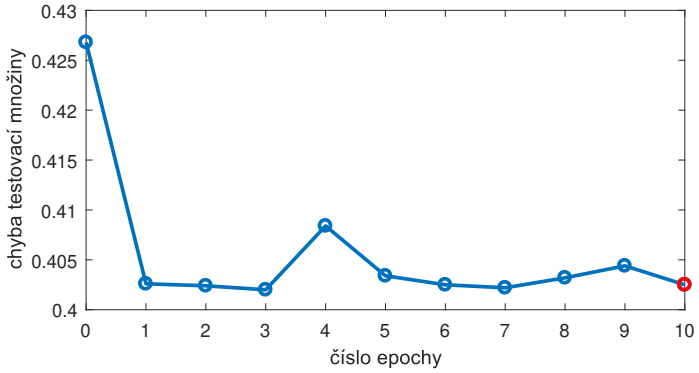
Obrázek 3.10: Architektura výsledné sítě[7]

3.5.3 Průběh učení sítě

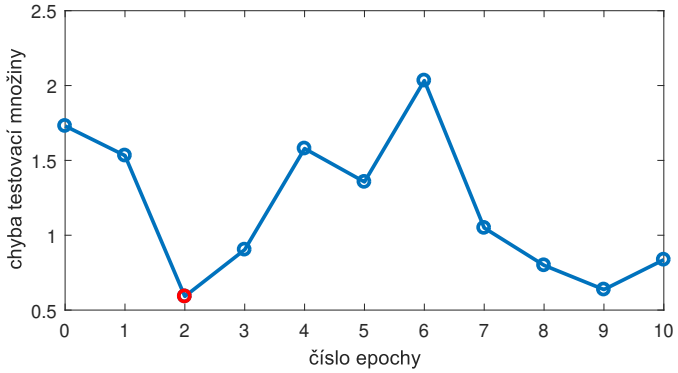
Výslednou síť jsme učili ve 2 vlnách. Doba učení v každé vlně byla rozdělena na 10 epoch o stejném počtu iterací. Po každé epoše jsme vypočítali chybu na celé testovací množině a uložili váhy. Použili jsme ztrátovou funkci Cross entropy loss a parametry optimalizovali pomocí algoritmu Adam. Před začátkem trénování jsme použili Xavier inicializaci vah sítě. Nejprve jsme učili síť na celé trénovací množině s vyšší rychlostí učení ϵ. Vybrali jsme váhy s nejlepšími výsledky na testovací množině. Při vizualizaci výsledků se ukázalo, že má síť problémy s rozpoznáváním nákladních automobilů, proto jsme ji nechali učit na množině s jejich vyšším zastoupením s nižší rychlostí učení ϵ. Opět jsme vybrali verzi sítě s nejnižší testovací chybou. Epochy s minimální chybou, s jejichž vahami jsme dále pracovali, jsou vyznačeny v grafech červenou barvou.



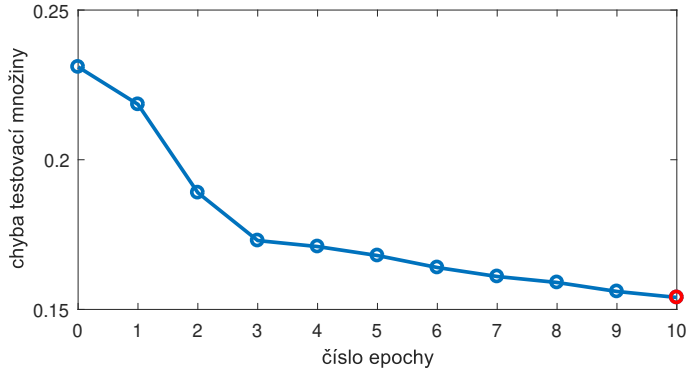
Obrázek 3.11: Graf průběhu testovací chyby pro 2 kategorie, 1. vlna



Obrázek 3.12: Graf průběhu testovací chyby pro 2 kategorie, 2. vlna



Obrázek 3.13: Graf průběhu testovací chyby pro 3 kategorie, 1. vlna



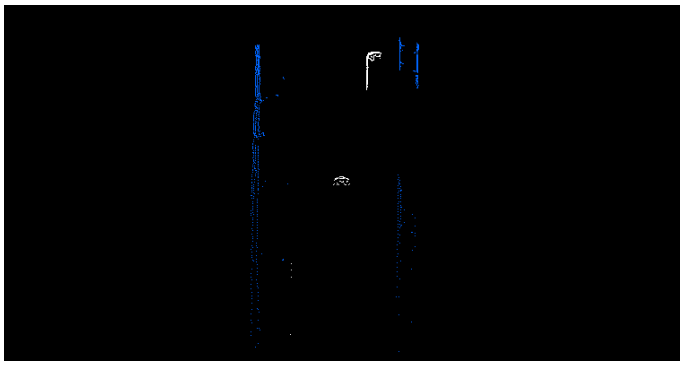
Obrázek 3.14: Graf průběhu testovací chyby pro 3 kategorie, 2. vlna

3.5.4 Výsledky segmentace

Testovali jsme 2 varianty sítě. První z nich rozpoznává 3 kategorie, tedy pozadí, auta a nákladní automobily. Druhá rozlišuje pouze 2 kategorie, osobní a nákladní automobily jsou sdruženy do jedné.

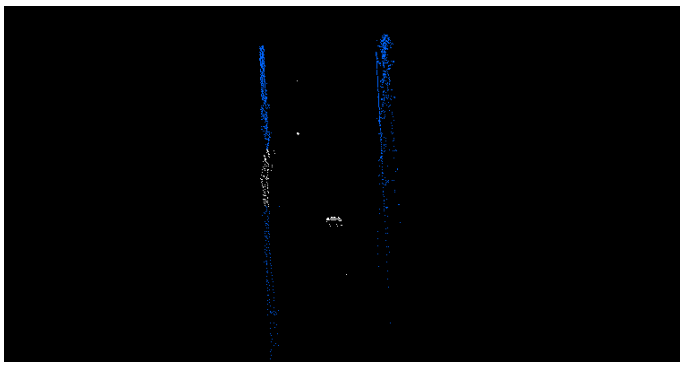
2 kategorie

V případě 2 kategorií stačí správně rozpoznat okraje silnice a označit body mezi nimi za vozidlo. Síť tedy ve většině případů dosahuje velmi dobrých výsledků, jak můžeme vidět na obrázku 3.15.



Obrázek 3.15: Příklad úspěšné predikce

V některých případech ale síť zamění rovnou část svodidel za bok nákladního automobilu a chybně ji klasifikuje. Příklad tohoto jevu vidíme na obrázku 3.16.



Obrázek 3.16: Příklad chybné predikce

Po segmentaci všech prvků trénovací množiny o přibližně 1600 prvcích jsme dosáhli výsledků zaznamenaných v matici záměn.

		Reference	
		pozadí	**vozidlo**
Predikce	**pozadí**	1049773	9507
	vozidlo	35544	105607

Tabulka 3.6: Matice záměn pro segmentaci 2 kategorií

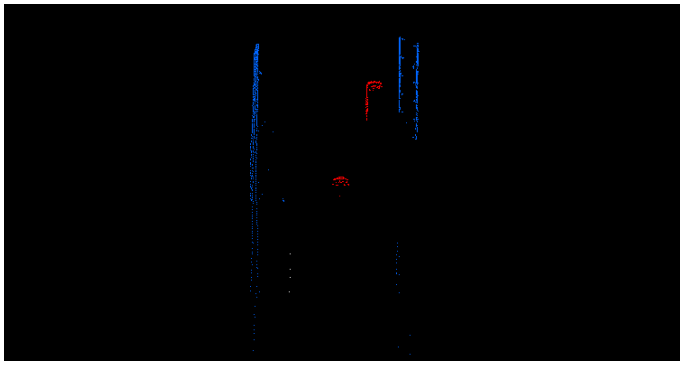
Z matice záměn určíme hodnoty precision a recall:

	precision	**recall**
pozadí	99.1%	96.7%
vozidlo	74.82%	91.7%

Tabulka 3.7: Tabulka hodnot precision a recall pro 2 kategorie

3 kategorie

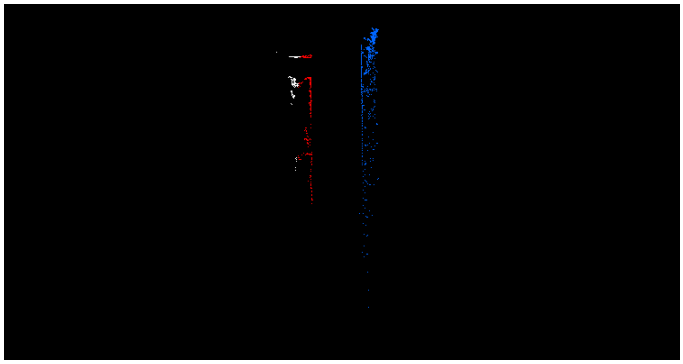
Protože bylo v trénovací množině velké množství aut, síť nemá problém s jejich rozpoznáním. Na obrázku 3.17 vidíme výsledek takové predikce, kde síť správně označila 2 auta.



Obrázek 3.17: Příklad úspěšné predikce

Problém nastává u segmentace nákladních automobilů, kterých bylo v

trénovací málo. Síť správně rozpozná jejich část, ale zbytek bodů klasifikuje jako auto. Čím blíže je vozidlo k senzoru, tím se tato chyba více projeví, protože síť nemůže kategorie rozlišit podle jejich výšky.



Obrázek 3.18: Příklad chybné predikce

Výsledky na celé testovací množině můžeme vidět v matici záměn:

		Reference		
		pozadí	**auto**	**nákl. auto**
Predikce	**pozadí**	1062295	26570	14
	auto	4334	41788	8667
	nákl. auto	18688	3759	34316

Tabulka 3.8: Matice záměn pro segmentaci 3 kategorií

Veličiny precision a recall mají hodnotu:

	precision	**recall**
pozadí	97.56%	97.9%
auto	76.27%	57.9%
nákl. auto	60.45%	79.8%

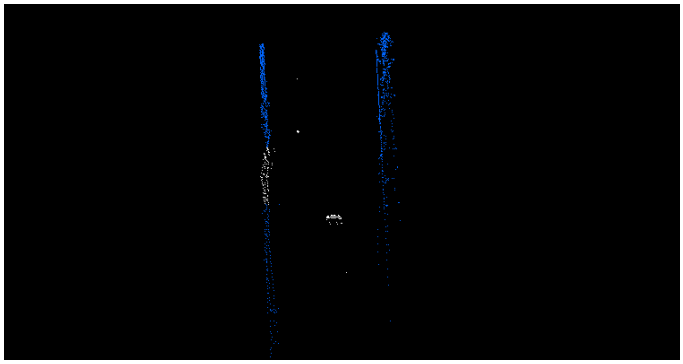
Tabulka 3.9: Tabulka hodnot precision a recall pro 3 kategorie

3.5.5 Postprocessing výsledků

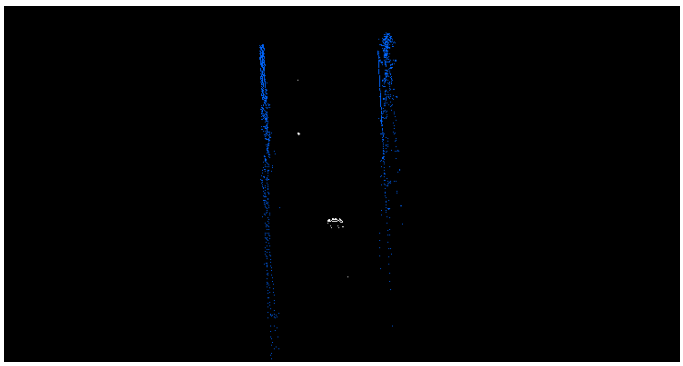
Výsledky segmentace obvykle obsahují chyby, které se můžeme pokusit dodatečně opravit. Často se jedná o opakující se chyby, například může síť zaměnit rovnou část svodidel za bok nákladního automobilu. Použili jsme způsob filtrace, při kterém projíždíme predikovanou matici vhodně zvoleným jádrem. V případě filtrace svodidel vybereme úzké svislé jádro o rozměru 80×5, které dobře kopíruje úzký pás bodů svodidel. Pokud se v dané podoblasti matice nachází většina bodů náležících pozadí, označíme všechny body v podoblasti jako pozadí. Podobným způsobem můžeme postupovat pro všechny kategorie. Je důležité správně vybrat velikost jádra pro každou kategorii tak, aby zbytečně nedocházelo k chybným úpravám.

2 kategorie

V případě 2 kategorií jsme se snažili odstranit označování svodidel za vozidlo. Na obrázku 3.19 vidíme typický příklad takové chyby. Filtrovali jsme tedy s prioritou pozadí s jádrem o velikosti 80×5. Obrázek 3.20 ukazuje výsledek filtrace se správně označenými svodidly. Tato metoda nedokáže odfiltrovat izolované body pozadí ležící mezi pásy svodidel. V případě chybné klasifikace vozidla, kdy je jeho část označena za pozadí může filtrace tuto chybu ještě prohloubit.



Obrázek 3.19: Predikce před filtrací



Obrázek 3.20: Predikce po filtraci

Po filtraci celé testovací množiny dostáváme výsledky zobrazené v matici záměn.

		Reference	
		pozadí	**vozidlo**
Predikce	**pozadí**	1058921	10027
	vozidlo	26396	105087

Tabulka 3.10: Matice záměn pro 2 kategorie, po filtraci

U kategorie vozidlo pozorujeme po filtraci nárůst precision a mírný pokles recall.

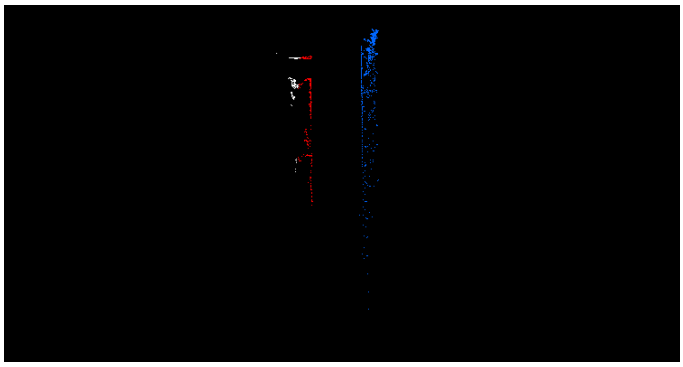
	precision	**recall**
pozadí	99.06%	96.6%
vozidlo	79.92%	91.3%

Tabulka 3.11: Tabulka hodnot precision a recall pro 2 kategorie, po filtraci

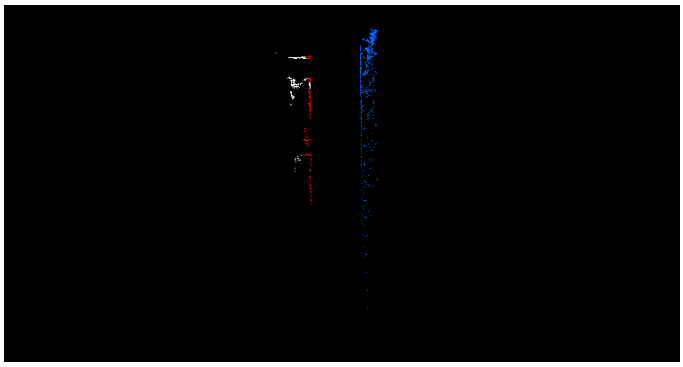
3 kategorie

U predikce 3 kategorií se nejčastěji potýkáme s problémem chybné segmentace části nákladního automobilu označené jako auto. Příklad tohoto jevu můžeme vidět na obrázku 3.21. V tomto případě uplatníme filtraci s prioritou nákladních automobilů s jádry o rozměrech 80×5 a 5×20 pro svislé stěny vozidla

a jeho čelo. Filtrace není tak úspěšná jako v případě 2 kategorií, podařilo se nám alespoň snížit chybu predikce, viz obrázek 3.22.



Obrázek 3.21: Predikce před filtrací



Obrázek 3.22: Predikce po filtraci

Výsledky filtrace celé testovací množiny jsou vyneseny v matici záměn.

		Reference		
		pozadí	**auto**	**nákl. auto**
Predikce	**pozadí**	1062562	27184	14
	auto	4012	40707	6462
	nákl. auto	18743	4226	36521

Tabulka 3.12: Matice záměn pro 3 kategorie, po filtraci

Podařilo se nám u kategorie nákladní auto zvýšit recall a mírně vylepšit i precision. K mírnému zlepšení došlo i u kategorie osobních automobilů. Tam narostla precision, bohužel na úkor recall.

	precision	recall
pozadí	97.5%	97.9%
auto	79.54%	56.4%
nákl. auto	61.39%	84.9%

Tabulka 3.13: Tabulka hodnot precision a recall pro 3 kategorie, po filtraci

3.6 Klasifikace pohybujících se objektů

Se segmentovanými daty můžeme dále pracovat. Protože se jedná o sekvenci snímků, je možné využít znalost predikce předešlých měření. Můžeme se například pokusit rozeznat jednotlivá vozidla a odhadovat jejich pohyb napříč snímky. Tato informace je velmi cenná pro autonomní řízení, protože nám umožňuje předvídat pohyb okolních vozidel. Tento úkol můžeme rozdělit do tří dílčích částí: hledání jednotlivých instancí vozidel, nalezení příslušného páru instancí v po sobě jdoucích snímcích a odhad rychlosti a směru pohybu vozidla. Pro tuto úlohu se omezíme pouze na segmentaci 3 kategorií.

3.6.1 Hledání instancí vozidel

Úlohu hledání jednotlivých vozidel můžeme formulovat jako hledání lokálních maxim hustoty výskytu dané kategorie. Nejdříve vytvoříme pro danou kategorii binární mapu, ve které 1 značí predikci hledané kategorie a 0 značí jiný výsledek. Následně provedeme konvoluci s jádrem o velikosti hledaného objektu, v případě auta použijeme jádro o rozměru 20×50, pro nákladní automobil 30×180. Poté provedeme ještě vyhlazovací konvoluci s jádrem 5×5 a ve výsledné mapě hustoty hledáme maximum. Po nalezení maxima si zaznamenáme jeho polohu, vynulujeme jeho okolí a pokračujeme v hledání. V případě že hodnota nalezeného maxima je nižší než předem definované minimum, hledání končí. Dále pro zpřesnění výsledku hledáme nejmenší podoblast jádra obsahující všechny body nalezeného objektu a dále pracujeme se středem této podoblasti.

3.6.2 Hledání párů instancí

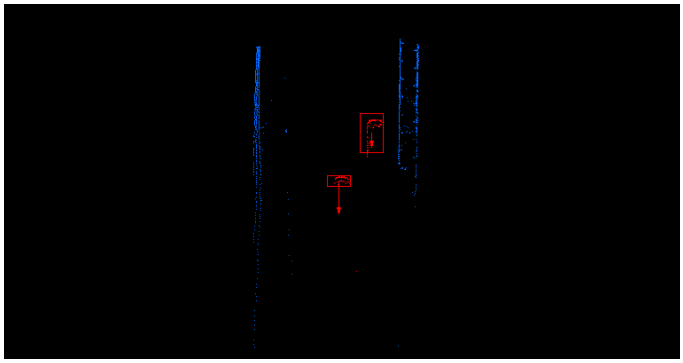
Po nalezení středů vozidel v po sobě jdoucích snímcích je třeba určit dvojice náležící stejnému vozidlu. Polohy vozidel napříč snímky se tedy výrazně neliší, protože se jedná převážně o snímky z dálnic s malými rozdíly v rychlosti vozidel. Díky tomu můžeme hledat páry instancí jako dvojice středů s minimální vzájemnou vzdáleností. Podobně jako při hledání středů nejprve najdeme dvojici s nejmenší vzdáleností, zaznamenáme si ji, odstraníme ze seznamu a hledáme znovu. Pokud je vzdálenost vyšší, než námi určená maximální hodnota, ukončíme hledání.

3.6.3 Odhad pohybu

Při odhadu pohybu nalezeného vozidla musíme uvažovat i pohyb měřícího automobilu. Ten určíme jako rozdíl středů poloh senzoru v jednotlivých měřeních. Informace o poloze a natočení senzoru ve vztahu k začátku měření jsou k dispozici ve formě transformačních matic. Výsledný pohyb vozidla je tedy vektorovou kombinací relativního pohybu napříč snímky a pohybu senzoru.

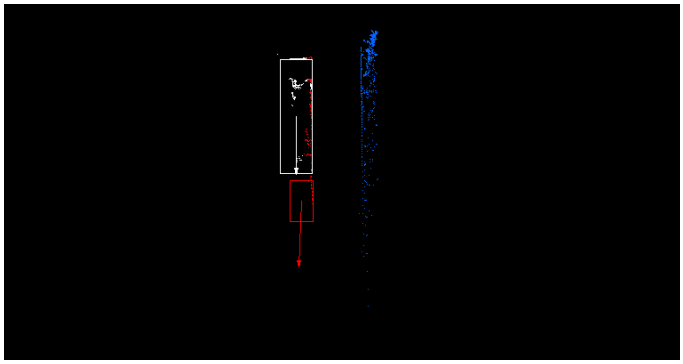
3.6.4 Výsledky

Pro odhad pohybu nemáme žádné referenční hodnoty, výsledky tedy musíme vyhodnotit bez nich. Úspěšnost odhadu pohybu je závislá na úspěšnosti predikce. V případě správné segmentace můžeme dosáhnout velmi dobrých výsledků. Na obrázku 3.23 vidíme 2 auta s různou rychlostí, znázorněnou délkou šipky. Rychleji jedoucí levé auto předjíždí pomalejší pravé auto, což byl algoritmus schopen správně odhadnout.



Obrázek 3.23: Ukázka správného odhadu směru

V případě chybné predikce nastává několik problémů. Na obrázku 3.24 můžeme vidět část nákladního automobilu klasifikovanou sítí jako auto. Tuto část označil algoritmus za instanci auta a rozdělil tak objekt na 2 části, ve skutečnosti ale všechny body náleží nákladnímu automobilu. Dalším problémem je chybné určení rychlosti. Pohyb vozidla odhadujeme podle středu shluku bodů, které jej určují. Při chybné predikci se ale mění velikost chybně označené části vozidla a tedy i námi určený střed.



Obrázek 3.24: Ukázka nesprávného odhadu směru

Kapitola 4

Závěr

Cílem této práce bylo navrhnout konvoluční neuronovou síť pro klasifikaci LiDARových dat, představujících jedoucí automobily a dopravní situace. Vyzkoušeli jsme 2 různé reprezentace dat, hloubkové mapy a obrázky z ptačí perspektivy. Pro každou z těchto reprezentací jsme navrhli a natrénovali konvoluční síť. Sítě jsme trénovali nejprve na simulovaných datech ze hry *Grand Theft Auto V* a později i na reálných datech od firmy Valeo.
V případě sítě pracující s hloubkovými daty jsme nedosáhli dobrých výsledků znázorněných v matici záměn. Dle našich experimentů se tato reprezentace neosvědčila pro segmentaci pomocí neuronových sítí. Naproti tomu se osvědčilo použití obrázků z ptačí perspektivy. Pro ně jsme vytvořili 2 sítě, rozpoznávající 2 a 3 kategorie. Síť rozlišující 2 kategorie (pozadí, vozidlo) byla úspěšnější než síť rozlišující 3 kategorie (pozadí, osobní automobil, nákladní automobil), protože měla jednodušší úlohu. Procentuální přesnost predikce jsme byli schopni ještě vylepšit automatickou filtrací.
Tyto výsledky dále zpracováváme a z jejich sledu odhadujeme pohyb detekovaných vozidel. Úspěšnost těchto odhadů je závislá na kvalitě predikce.
Do budoucna by bylo možné zdokonalit výsledky predikce kvalitnější filtrací vstupních dat, která by odstranila body náležící silnici, které síť často není schopna správně klasifikovat. Dále by bylo možné vyzkoušet jiné architektury sítě. Úspěšnost odhadu pohybu by bylo možné zvýšit použitím více po sobě jdoucích predikcí nebo dokonalejším způsobem určení středů instancí vozidel.

Příloha A

Literatura

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts, 2016. URL https://www.deeplearningbook.org/.

[2] Jayesh Bapu Ahire. *Artificial neuron*. URL https://cdn-images-1.medium.com/max/800/1*WRG_Re8vGVuHDYigtq2IBA.jpeg.

[3] An example artificial neural network with a hidden layer, December 2006. URL https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg.

[4] A technical report on convolution arithmetic in the context of deep learning: vdumoulin/conv_arithmetic, May 2019. URL https://github.com/vdumoulin/conv_arithmetic. original-date: 2016-02-24T15:18:33Z.

[5] *Valeo Scala® : a laser scanner for highly automated driving*. . URL https://www.valeo.com/en/valeo-scala/.

[6] *CloudCompare - Open Source project*. . URL https://www.danielgm.net/cc/.

[7] Alexander LeNail. *NN SVG*. URL http://alexlenail.me/NN-SVG/LeNet.html.

[8] *Valeo Scala datasheet*. . URL https://autonomoustuff.com/wp-content/uploads/2018/04/ibeo_ScaLa_datasheet_v2.pdf.

[9] *Velodyne HDL-64E datasheet*. . URL https://velodynelidar.com/lidar/products/brochure/HDL-64E%20S2%20datasheet_2010_lowres.pdf.

[10] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5 (4):115–133, December 1943. ISSN 1522-9602. doi: 10.1007/BF02478259. URL https://doi.org/10.1007/BF02478259.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

[12] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning Deconvolution Network for Semantic Segmentation. *arXiv:1505.04366 [cs]*, May 2015. URL http://arxiv.org/abs/1505.04366. arXiv: 1505.04366.

[13] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. URL http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.

[14] Bin Yang, Wenjie Luo, and Raquel Urtasun. PIXOR: Real-time 3d Object Detection from Point Clouds. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, Salt Lake City, UT, USA, June 2018. IEEE. ISBN 978-1-5386-6420-9. doi: 10.1109/CVPR.2018.00798. URL https://ieeexplore.ieee.org/document/8578896/.